Remarks/Arguments:

The above amendment modifies the specification in direct response to requests for correction in the 3/30/2007 Office Action (OA). The amendment defines acronyms prior to their use, and makes the abstract more concise, as requested in the OA.

Additionally, the amendment replaces the claims to define the invention more specifically to overcome the objections under 35 U.S.C. 102(e) of the OA, and to be patentable over prior art. Specifically of the original Claims 1-19:

- Claims 1, 7, 9-14, 17, and 18 have been canceled.
- Claims 2, 3, and 8 have been amended, and are now independent claims.
- Claims 4-6, 15, 16, and 19 have been amended, and continue to be dependent claims.
- Claim 20 is a new claim, which is dependent on Claim 8 and includes material similar to the original Claim 11.

Summary of References

The OA includes the following two references:

Omoigui, US Patent Publication No. 2003/0126136, discloses a system and method for knowledge retrieval, management, delivery and presentation. This reference is cited throughout the OA objections to the claims. Omoigui *"adds three additional platform layers not present in Today's Web: Knowledge Indexing and Classification Layer, wherein information from*

both structured and unstructured sources are semantically encoded; Knowledge Representation Layer, wherein associations are created that allows maintenance of a self-correcting or healing Semantic Network of knowledge objects; and Knowledge Ontology and Inference Layer, wherein new connections and properties are inferred in the Semantic Network." Both overall and specific differences between Omoigui and the currently amended claimed invention will be discussed below.

Prakash, US Patent Publication No. 2002/0123984, discloses a framework for dynamic query of server applications. This reference was not cited within the objections to the claims, but was noted by the OA as pertinent. Like Omoigui, Prakash proposes query-building and user-interface facilities layered on top of existing web applications. The applicant reviewed this reference, and did not find it to show the amended claims, nor to render them obvious.

Note that in the current application, the word *specification* is used to mean a collection of *clauses* (also known as *conditions* or *constraints*) which specify desired computational results. In the references the word *query* is used to mean substantially the same thing. The following remarks will use "*specification*" and "*query*" interchangeably.

Omoigui Adds Three new Layers to Three-Tier Model, While The Invention Unifies Three-Tier Model

Before discussing how each amended claim overcomes the OA objections, we will summarize the differences from Omoigui. The central insight for the current invention was to unify and replace the traditional three layers of web application development (which are a declarative presentation

specification layer, a procedural business logic layer, and a structured or unstructured data layer). The unification eliminates the need for separate description languages for each layer, and for troublesome integration between different paradigms at each layer. Omoigui, on the other hand, is an application architecture which adds three new layers to the three traditional layers, and does not anticipate the current invention nor reap its advantages. Omoigui does add novel and useful structure, including declarative logic components and semantic content and search. The current invention is an improved alternative architecture and toolkit with which to build applications like Omoigui.

The further remarks below detail how each active claim, as amended, defines novel and unobvious invention.

Specific Method of Defining Arbitrary Aggregations in Claim 2 is Novel

The amended Claim 2, in paragraph (k), defines a specific way in which aggregating clauses are defined: they *specify a further set of variable values in a current iteration based on the variable values in the previous iteration*. For example, a product of results could be specified with the XSP clause:

```
<xsp:multiply in="1" var:by="amount" var:sum="product"/>
```

Thus in the invention, aggregating clauses can specify any arbitrary calculation over a series of iterations.

In the references cited in the OA ([0865], [0899], [0974]) Omoigui does not specify how aggregation is to be done, though it does imply that one of Omoigui's "SQML" queries may combine queries to different servers, and in

this sense "aggregate" by presenting a conclusion based on separate sources, but does not define any mechanisms in SQML for specifying calculated summary aggregations, much less the mechanism of Claim 2.

Omoigui also mentions translation of queries into SQL, XQL, and XQuery, which have mechanisms to define aggregation. These mechanisms, however, do not provide for an arbitrary calculation of result values from an iteration. SQL, XQL, and XQuery provide special predefined aggregates, including summation, maximum, minimum, and counting aggregates. They do not provide a declarative mechanism for the user to create these or other aggregates.

Aggregation Mechanism of Claim 2 is Unobvious

Unlike a procedural or imperative language, which describes what steps comprise an algorithm and how system memory should be changed at each step, a functional or declarative language specifies attributes of the desired result. Many authors have discussed the benefits of declarative, side-effect free languages. They are known to be easier to verify, optimize, and parallelize (see the "functional programming" article on Wikipedia, for example). With a declarative language, the computer can find a more-optimal algorithm at run-time to achieve the desired result, can determine facts about the result of the algorithm without executing it, can run portions of the program concurrently on different processors, and can prove program correctness.

As discussed above, the aggregation mechanism of Claim 2 is a novel way to define aggregates within a declarative query language. As evidence that this mechanism is also unobvious, note that the providers of the SQL

language engines, which Omoigui is built upon, have continued to add new predefined aggregates (for example a "moving average" aggregate in Oracle's database version 8i), to compete in the marketplace. Nevertheless, these companies did not provide users a simple way to define arbitrary new aggregates, in spite of the obvious benefit this would provide. (For example, Oracle's version 9i introduced a cumbersome procedural add-on, separate from declarative SQL, to define custom aggregates.)

The flexible mechanism for specifying arbitrary aggregation of Claim 2, is necessary to make the declarative language *"Turing-Complete"*, which means that it can specify any computational algorithm that can be specified with any other "Turing-Complete" language, like Java or C. Many algorithms cannot be specified in other non-declarative languages like SQL. As illustration, note that vendors of declarative languages like SQL have resorted to allowing users to integrate bits of Java and C-language algorithms into their SQL-based systems, or defined special non-declarative languages like PL-SQL which are used for additional functionality.

Thus the unexpected benefit of Claim 2's aggregation mechanism is to provide a declarative language, with the associated benefits (easier to verify, optimize, and parallelize), that does not require an adjunct procedural language which would necessarily limit or eliminate those benefits.

Because the described generation mechanism, as amended, is novel and unobvious, Claim 2 should be allowed.

Backtracking Distinguishes Claim 3

The examiner correctly cited Omoigui ([0971], [1139]) as describing an

ordering of clauses similar to Claim 3. However the *backtracking and re-evaluating an antecedent clause to generate alternate sets of variable values when a subsequent clause fails* (l) structure of Claim 3, as amended, is not disclosed in [0973] nor elsewhere in Omoigui. Omoigui describes a recursive mechanism by which each SQML resource is expanded into other SQML resources. This process does not allow for backtracking, in which an antecedent resource would be re-expanded in a different way if a subsequent resource fails. This backtracking is unobvious. As evidence, note that if Omoigui had included it, it would allow the unexpected efficiency benefit of finding a desired combination of antecedent and subsequent query results without having to retrieve all results of the antecedent query.

Because Claim 3, as amended, is novel and unobvious, it and the dependent claims 4, 5, 6, 15, 16, and 19 should be allowed.

Specific and Novel Planning Methods Distinguish Claims 4 and 5

Claims 4 and 5, as amended, describe two novel and beneficial elements of a particular mechanism to plan the execution of a query. This mechanism uses two phases to produce a procedural (step-by-step) execution plan from a declarative specification (unordered list of constraints):

1. In the first phase, a subset of the clauses in the query specification are selected to "generate" query variables.

2. In the second phase, all the clauses are ordered in a way that will optimize execution.

The following remarks will detail how each of these phases are novel and

unobvious, and overcome objections to the original Claims 4 and 5.

Objection to Claim 4 Overcome

The cited sections of Omoigui ([1117], [1125]), disclose allowing the user to specify certain numerical conditions on acceptable result documents, for example the number of articles in an email "thread", or minimum and maximum acceptable ages for a document. Omoigui does not disclose using these or any other numeric attributes for ordering the steps of a search or computation. Omoigui also does not disclose a comparable two-phase algorithm for planning, or any other particular algorithm for planning, other than to translate queries into the well-known "SQL" language for execution.

Claim 4 is the first step of the two-step planning phase. This first step picks which clauses will be used to generate ("produce") variable values. All other clauses which mention this value will simply use ("consume") it. The specific numeric attribute used for selection in Claim 4 is the number of distinct variable value results ("branching"). Further, this branching is re-evaluated after each selection of a clause in the planning process. This produces the unexpected benefit of dramatically increasing the efficiency of the final plan (because it can iterate over a smaller number of possible values), and of providing additional opportunity for optimization in a second ordering phase (because the clauses can still be re-ordered) without inordinate computation during the planning phase (because we do not have to consider every possible ordering of the clauses). Thus the "pick next generating clause based on minimum branching" rule of Claim 4 is both novel and unobvious.

Objection to Claim 5 Overcome

The further cited sections of Omoigui ([0072], [0322], [0341]) mention the ability to optimize user input, presentation, and retrieval. None of the specific mechanisms for this optimization are described, save for the use of "natural language" processing for information retrieval. None of these disclose the particular numeric attribute and ordering process described in Claims 4 and 5. Using a numeric attribute based on both cost and branching provides the unexpected benefit of efficient computation (because clauses that are more expensive to compute are executed less often, because they are executed before branching can cause additional backtracking and re-evaluation), while remaining also efficient to plan.

Because each phase of the described two-phase planning method is novel and unobvious, Claims 4 and 5 as amended, and the dependent claims 6, 15, 16 and 19 should all be allowed.

Dynamic Dispatch Applied to Semi-Structured Data Distinguishes Claim 6 and Claim 19

Omoigui ([0865], [0899]) discloses methods of following semantic links, where the SQML eventually returned may depend on the dynamic value of a variable object. Claim 6 and Claim 19, as amended, specify a particular method for "object-oriented" dynamic dispatch, in which the labels of semi-structured data values (e.g. the element tag name and namespace of an XML element) are used to determine a location to look for a clause definition. Omoigui does not disclose any dispatch based on the type of an object, much less the specific use of the labels of semi-structured data for this purpose. (Omoigui dispatches based on object storage location).

While object-oriented dynamic dispatch is common in computer programming languages, Claim 6 and Claim 19 represent a novel technique for applying object-oriented dispatch to semi-structured data. This technique unexpectedly provides the benefits of object-oriented dispatch (e.g. polymorphism and inheritance) without requiring the addition of any type information beyond that already present in semi-structured data.

Thus Claim 6 and Claim 19, as amended, are novel, the technique unobvious, and should be allowed.

Method of Making Data Changes Compatible with Generate-and-Test and User-Specified Clauses Distinguishes Claims 8 and 15

Claims 8 and 15, as amended, both define a specific method for combining declarative specifications with clauses, specifying data changes with some of the clauses, and implementing generate-and-test for evaluation of those specifications. This particular combination is novel and unobvious, as discussed below.

Omoigui discloses a number of ways in which data storage can be changed, including via email messages, via SQL-based databases to store cached source data, and via the storage and updating of discovered semantic data. Omoigui also discloses a declarative language, SQML, which specifies desired results and an evaluation of those results by translating to a database query language like SQL. However, Omoigui does not disclose any method, much less the method of Claims 8 and 15, to specify arbitrary data changes within SQML. Thus Claims 8 and 15 are not disclosed by Omoigui.

In traditional data management systems, changes to data are described by

the user in either

1. a purely procedural manner (e.g. step 1: set x=3; step 2: store x at location a; step 3: read y from location b...), or

2. a declarative if-then rule (e.g. IF x=3 and location a contains 4 THEN store x at location b.).

The first example is in the style of procedural languages like "C", "Perl", and the second example is in the style of an SQL database (which uses UPDATE to indicate the "THEN" portion, and WHERE to indicate the "IF" portion of the rule). The data changes caused by execution of these descriptions can be called "side-effects" of the execution.

A procedural language allows for user-specified clauses and for data changes, but does not provide generate-and-test, since statements can not be "undone" and "redone". This lack of generate-and-test means that the system cannot automatically search for desired results, rather the user must specify the step-by-step algorithm to produce the results. Thus Claim 8 and 15 are not anticipated by any procedural languages, including those mentioned by Omoigui.

A traditional declarative language, e.g. for an SQL database, allows for generate-and-test search through the database, which occurs as the database searches through the data for combinations that satisfy the "IF" or "WHERE" clause. An SQL database may provide for user-defined clauses, however these clauses must be either wholly free of any data changes (e.g. database "views"), or are procedural units that can only be included in the "THEN" portion of a definition (e.g. "stored procedures"). A traditional declarative language does not allow for user-defined clauses to

include data side-effects and be used within the powerful generate-and-test engine. Thus traditional declarative languages, including the ones mentioned by Omoigui, do not anticipate Claim 8 or 15.

That the mechanism of Claim 8 and 15 is unobvious is demonstrated by:

a) it is contrary to conventional design of either procedural or declarative systems,

b) it has not been implemented in Omoigui or anywhere else to the knowledge of the applicant,

c) it provides the benefit of using a single language to describe both arbitrary data changes and arbitrary data searches, and

d) it allows the intermixing of side-effects and data queries into reusable component clauses.

Thus Claims 8 and 15 as amended are both novel and unobvious, and Claims 8 and 15 and the dependent Claims 16, 19, and 20 should be allowed.

Method of Updating Indexes Distinguishes Claims 16 and 20

Omoigui discloses caching of intermediate query results and indexing information by meta data and semantics. Also SQL databases are well-known to index columns of data tables. The caching and indexing method claimed in both Claim 16 and Claim 20 is different, because it specifies a novel means for updating the indexes when underlying data changes.

Omoigui does not directly describe any mechanism for updating indexes, much less the specific mechanism of Claims 16 and 20. Traditionally, query

systems will simply put a time and date on cached results, and recalculate the results after a set period of time. One disadvantage to this method is that if the data changes during the set period, the cache becomes "stale", and invalid results may be returned. The other disadvantage is that if the data has not changed, then an unnecessary recalculation is done after the set period expires.

SQL and other databases, mentioned in Omoigui, also have various caching and indexing mechanisms, typified by either a *column index*, a *query plan cache*, or a *query result cache*. We will discuss each in turn in order to contrast with the invention:

- A column index is efficiently partially recalculated when specific rows of a data table are changed. The limitation of a column index is that it can only cache values of a particular column, not of an arbitrary query.

- A query plan cache stores execution plans for queries, so that subsequent uses do not have to plan again. The limitation of these caches is that the query plan must be re-executed each time the query result is needed.

- A query result cache may store the full result of a query. The limitation of a query result cache is that either it is not updated when data changes, causing potentially stale results, or an entire cache associated with a particular table must be (inefficiently) recalculated when even the smallest portion of the data in the table is changed.

The mechanism of Claims 16 and 20 is novel, because when data changes, it uses a re-execution of a modified version of a query plan to discover which

portions of the cache and index need to be modified. This novelty is also unobvious, as demonstrated by the unexpected benefits that it:

a) eliminates the risk of stale results being returned,

b) allows arbitrary queries to be cached for quick execution, and

c) minimizes the amount of updates required to the cache and index.

Thus Claim 16 as amended, the dependent claim 19, and the new Claim 20, are novel and unobvious, and should be allowed.

Conclusion

For all the above reasons, applicant submits that the specification and claims are now in proper form, and that the active claims all define patentably over the prior art. Therefore he submits that this application is now in condition for allowance, which action he respectfully solicits.

Conditional Request for Constructive Assistance

Applicant has amended the specification and claims of this application so that they are proper, definite, and define novel structure which is also not obvious. If for any reason this application is not believed to be in full condition for allowance, applicant respectfully requests the constructive assistance and suggestions of the Examiner pursuant to M.P.E.P. Section 2173.02 and Section 707.07(j) in order that the undersigned can place this application in allowable condition as soon as possible and without the need for further proceedings.

Very respectfully,

Judson Ames Cornish
646 Georgia Avenue
Palo Alto, CA 94306
650-533-0835 (Mobile)
650-331-1402 (Daytime)
650-494-2639 (Evenings)
815-301-9795 (Fax)
ames@cornishes.net

CERTIFICATION

I certify that the Amendment C does not include any new matter. I further certify that this correspondence and attachments will be deposited with the United States Postal Service by First Class Mail, postage prepaid, in an envelope addressed to Commissioner for Patents, PO Box 1450, Alexandria, VA 22313-1450, on September 29, 2007.

Applicant: _____

Date: 9/28/2007